Statistical Techniques in Robotics (16-831, S21) Lecture #05 (Monday, February 15)

OLC (Perceptron, Winnow)

Lecturer: Kris Kitani *Scribes: Xuhua Huang, Xinjie Yao*

1 Review

In the last lecture, without the assumption of at least one perfect expert (realizability), we dive into two main algorithms: (1) Weighted Majority Algorithm (WMA), and (2) Random Weighted Majority Algorithm (RWMA) [3]. In this lecture, we will shift to a new topic named Online Linear Classification (OLC). One common feature between OLC and WMA is that they only have access to a subset of states, so they are both sampled learning. We will introduce and analyze two approaches to learn an online linear classifier: Perceptron Algorithm [7] and Winnow Algorithm [2]. As a high-level takeaway, both of them will construct a linear classifier on-the-fly, while the key differences are: (1) weight update rules (additive updates vs. multiplicative updates) (2) prediction function (hyperplane classifier vs. disjunctive boolean function). Before jumping into these two algorithms, we will recap WMA and RWMA together with a few definitions to understand the regret bound. Then, we will introduce the context of online linear classifiers with their problem setup.

1.1 Weighted Majority Algorithm(WMA)

WMA makes learner prediction based on a sum of weighted experts and updates those weights with a penalty parameter η. Specifically, the prediction rule and weight update rule are as follows,

$$\hat{y}^{(t)} = \text{sign}\Big(\sum_{n=1}^{N} x_n^{(t)} \cdot w_n^{(t)}\Big) \in \{-1, 1\} \tag{1}$$

$$w_n^{(t+1)} \leftarrow w_n^{(t)}\big(1 - \eta \cdot \mathbf{1}[y^{(t)} \neq x_n^{(t)}]\big) \tag{2}$$

Since there is no prefect expert in this setup, we utilize regret of the learner, $R^{(T)}(H)$, a cumulative loss for not following the best hypothesis in the hypothesis class H. The regret is proved to be bounded by

$$R(h_n) = M^{(T)} - m_n^{(T)} \leq (1 + 2\eta)m_n^{(T)} + \frac{2 \log N}{\eta} \tag{3}$$

We observe that WMA has bounded regret with the upper bound grow linear over time. however, it is *NOT* a no-regret algorithm. The upper bound of average regret over time can't converge to 0 as $T \to \infty$.

1.2 Random Weighted Majority Algorithm(RWMA)

By adding some randomization, we demonstrate it in RWMA that the number of mistakes will be cut by a half. Also, it turns out to be a no-regret algorithm. Specifically, the learner makes prediction by sampling from a multinomial distribution of the weights.

$$h \sim \textsc{Multinomial}(\mathbf{w}^{(t)}/\Phi^{(t)}), \text{ where } \Phi^{(t)} = \sum_{n=1}^{N} w_n^{(t)}$$
$$\hat{y}^{(t)} = h_i(\mathbf{x}^{(t)}) \tag{4}$$

By only modifying the prediction rule, RWMA could yield a much better performance. The expected regret is further bounded by,

$$\mathbb{E}[R] \leq \eta m_n^{(T)} + \frac{\log N}{\eta} \tag{5}$$

If we set $\eta = \frac{1}{\sqrt{T}}$, it turns out RWMA is a no-regret algorithm.

2 Summary

2.1 Perceptron Algorithm

Now we will start with the first algorithm - Perceptron algorithm, in Online Linear Classification (OLC). The algorithm details are listed as Algorithm 2 below. It is worthy of attention that, the perceptron algorithm makes prediction based on a sign function (line 4), and it applies an additive update after receiving answer from nature (line 6). These are the two major differences compared with Winnow algorithm introduced in Section 2.2.

Algorithm 1 Perceptron algorithm

```
1: $\mathbf{w}^{(1)} \leftarrow 0$                                              ▷ Weight initialization
2: for $t = 1, \cdots, T$ do
3:     Receive ($\mathbf{x}^{(t)} \in \mathbb{R}^N$)                          ▷ Receive expert predictions
4:     $\hat{y}^{(t)} = \text{sign}\Big(\langle \mathbf{w}^{(t)}, \mathbf{x}^{(t)} \rangle\Big)$      ▷ Make learner prediction
5:     Receive ($y^{(t)} \in \{-1, 1\}$)                        ▷ Receive actual answer
6:     $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + y^{(t)} \cdot \mathbf{x}^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$     ▷ Weight update
7: end for
```

There are some characteristics of perceptron algorithm:

- It is fast. Because the prediction function is a dot product, and the update is just a sum, simple computation is involved.
- It is **not** a large margin classifier. It only uses sign to make decision, so there is no notion of margin in the algorithm.
- It does **not** work for non-separable data. The decision boundary of perceptron algorithm is defined by a linear hyperplanes, so it only works when data is linearly separable.

Now we will derive the mistake bound below, and before that we will recap a lemma about dot product.

Lemma 1. *The dot product of two vectors is largest when they are parallel.*

Theorem 2. *(Perceptron Mistake Bound) Let R be the norm of observations, $R = \max_t \|x^{(t)}\|$, and γ be the margin of separability, $\gamma = \min_t y_t \langle \boldsymbol{w}^\star, \boldsymbol{x}^{(t)} \rangle$ where $\|\boldsymbol{w}^\star\| = 1$, the Perceptron mistake is bounded as follows,*

$$M \leq \frac{R^2}{\gamma^2}$$

Proof. Let's define the potential function as the squared l_2 norm of weight vector by,

$$\begin{aligned}
\Phi(t) = \|\mathbf{w}^{(t)}\|^2 &= \sum_n (w_n^{(t)})^2 \\
&= \|\mathbf{w}^{(t-1)} + y^{(t)} \cdot \mathbf{x}^{(t)}\|^2 \quad \text{(where a mistake is made)} \\
&= \|\mathbf{w}^{(t-1)}\|^2 + \|\mathbf{x}^{(t)}\|^2 + 2y^{(t)} \langle \mathbf{w}^{(t-1)}, \mathbf{x}^{(t)} \rangle \\
&\leq \|\mathbf{w}^{(t-1)}\|^2 + \|\mathbf{x}^{(t)}\|^2 \quad \because (y^{(t)} \langle \mathbf{w}^{(t-1)}, \mathbf{x}^{(t)} \rangle < 0 \text{ if a mistake}) \\
&\leq \|\mathbf{w}^{(t-1)}\|^2 + R^2 \quad (\text{Recall } R = \max_t \|x^{(t)}\|)
\end{aligned} \tag{6}$$

By induction to the base case using Eq. 6, one can conclude the potential function is upper bounded by,

$$\Phi(t) \leq \|\mathbf{w}^{(0)}\|^2 + M^{(T)} \cdot R^2 = M^{(T)} \cdot R^2 \quad (M^{(T)} \text{ is the number of mistakes}) \tag{7}$$

We have upper-bounded the potential function, now let's derive the lower bound with an intermediate potential function,

$$\left\langle \boldsymbol{w}^*, \boldsymbol{w}^{(t)} \right\rangle \tag{8}$$

Here, $\boldsymbol{w}^*$ denotes a perfect classifier, whose norm is required to be 1 (i.e. unit vector), and $\boldsymbol{w}$ denotes our classifier at time step t. We will start by deriving the upper bound and lower bound of this dot product.

Upper bound:

From Lemma 1, we know Eq. 8 becomes largest when $\boldsymbol{w}^*$ equals to the unit vector oriented in the same direction as $\boldsymbol{w}$, stated as:

$$\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} = \arg\max_{\boldsymbol{w}'} \langle \boldsymbol{w}', \boldsymbol{w} \rangle \tag{9}$$

Because $\boldsymbol{w}^*$ is also a unit vector, so the dot product of Eq. 8 will satisfy this inequality,

$$\left\langle \boldsymbol{w}^*, \boldsymbol{w}^{(T)} \right\rangle \leq \left\langle \frac{\boldsymbol{w}^{(T)}}{\left\|\boldsymbol{w}^{(T)}\right\|}, \boldsymbol{w}^{(T)} \right\rangle = \frac{\left\|\boldsymbol{w}^{(T)}\right\|^2}{\left\|\boldsymbol{w}^{(T)}\right\|} = \left\|\boldsymbol{w}^{(T)}\right\| \tag{10}$$

Therefore, Eq. 8 is upper-bounded by the norm of the hyperplane $\boldsymbol{w}$ at time step t.

Lower bound:

Let's recap the weights $\boldsymbol{w}$ update rule at each time step t here,

$$\mathbf{w}^{(t)} = \mathbf{w}^{(t-1)} + y^{(t)} \cdot \mathbf{x}^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}] \tag{11}$$

By applying dot product with perfect classifier $\boldsymbol{w}^*$ on both side, we can get,

$$\begin{aligned}\left\langle \boldsymbol{w}^*, \boldsymbol{w}^{(t)} \right\rangle &= \left\langle \boldsymbol{w}^*, \boldsymbol{w}^{(t-1)} \right\rangle + y^{(t)} \left\langle \boldsymbol{w}^*, \boldsymbol{x}^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}] \right\rangle \\ &= \left\langle \boldsymbol{w}^*, \boldsymbol{w}^{(t-1)} \right\rangle + y^{(t)} \left\langle \boldsymbol{w}^*, \boldsymbol{x}^{(t)} \right\rangle \quad (\because \text{perfect classifier make no mistake}) \\ &\geq \left\langle \boldsymbol{w}^*, \boldsymbol{w}^{(t-1)} \right\rangle + \gamma, \text{ where } \gamma = \min_t y_t \left\langle \boldsymbol{w}^*, \boldsymbol{x}^{(t)} \right\rangle \end{aligned} \tag{12}$$

Because $y^{(t)} \left\langle \boldsymbol{w}^*, \boldsymbol{x}^{(t)} \right\rangle$ denotes the distance from each data points to the hyperplane and γ is the margin, which is the smallest distance among all points.

By induction starting from based case $t = 0$,

$$\left\langle \boldsymbol{w}^*, \boldsymbol{w}^{(1)} \right\rangle \geq \left\langle \boldsymbol{w}^*, \boldsymbol{w}^{(0)} \right\rangle + \gamma \tag{13}$$

We can add a single γ on RHS at each time step and reach,

$$\begin{aligned}\left\langle \boldsymbol{w}^*, \boldsymbol{w}^{(T)} \right\rangle &\geq \left\langle \boldsymbol{w}^*, \boldsymbol{w}^{(0)} \right\rangle + M^T \cdot \gamma \\ &= M^T \cdot \gamma \quad (\because \boldsymbol{w} \text{ is initialized to 0 at } t = 0)\end{aligned} \tag{14}$$

<u>Combine</u>:

Now we have got the upper bound of Eq. 8 from Eq. 10 and the lower bound from Eq. 14. By combing,

$$M^T \cdot \gamma \leq \left\langle \boldsymbol{w}^*, \boldsymbol{w}^{(T)} \right\rangle \leq \left\| \boldsymbol{w}^{(T)} \right\| \tag{15}$$

We have,

$$M^T \cdot \gamma \leq \left\| \boldsymbol{w}^{(T)} \right\| \tag{16}$$

The above equation still holds when we apply a power of 2 for both sides, and we get,

$$\left(M^T \cdot \gamma\right)^2 \leq \left\| \boldsymbol{w}^{(T)} \right\|^2, \text{ where } \left\| \boldsymbol{w}^{(T)} \right\|^2 \text{is our defined potential function} \tag{17}$$

Now that we have successfully derived the upper bound and lower bound of potential function in Eq. 7 and Eq. 17, we can combine both to get the mistake bound of Perceptron,

$$\begin{aligned}\left(M^T \cdot \gamma\right)^2 &\leq M^{(T)} \cdot R^2 \\ \therefore M^{(T)} &\leq \frac{R^2}{\gamma^2}\end{aligned} \tag{18}$$

2.2 Winnow Algorithm

Now we will introduce the second algorithm in OLC, named Winnow algorithm. Winnow assumes only a subset of features are relevant for classification. Its predictor (line 4) works as a disjunction function, trying to learn set of weights on relevant features and make sum greater than N on positive examples. The weight update rule (line 6) is multiplicative.

Algorithm 2 Winnow algorithm

```
1: $\mathbf{w}^{(1)} \leftarrow \{1, ..., 1\}$                                    ▷ Weight initialization
2: for $t = 1, \cdots, T$ do
3:     RECEIVE $(\mathbf{x}^{(t)} \in \{0,1\}^N)$                          ▷ Receive expert predictions
4:     $\hat{y}^{(t)} = \mathbf{1}[\langle \mathbf{w}^{(t)}, \mathbf{x}^{(t)} \rangle > N]$       ▷ Make learner prediction
5:     RECEIVE $(y^{(t)} \in \{0,1\})$                                  ▷ Receive actual answer
6:     $w_n^{(t+1)} \leftarrow w_n^{(t)}(1+\beta)^{(y^{(t)}-\hat{y}^{(t)})\cdot x_n^{(t)}}$          ▷ Weight update
7: end for
```

Different from perceptron where weight decays, Winnow could increase or decrease weight depending on the type of mistake. Recall the weight update rule (line 7 in Algorithm 2),

$$w_n^{(t+1)} \leftarrow w_n^{(t)}(1+\beta)^{(y^{(t)}-\hat{y}^{(t)})\cdot x_n^{(t)}} \tag{19}$$

It shows that making a mistake on a positive example brings different gain to weights of relevant features than making a mistake on a negative example. The weight update rule is summarized in the following table.

True ($y^{(t)}$)	Prediction ($\hat{y}^{(t)}$)	Type of mistake	Gain ($y^{(t)} - \hat{y}^{(t)}$)	Weight update
0	1	mistake on negative	-1	decrease by $\frac{1}{(1+\beta)}$
1	0	mistake on positive	1	increase by $(1+\beta)$

Table 1: Winnow updates weights depending on the type of mistake.

Lemma 3. *Assume there is a relevant feature whose index is n, and its weight is w_n^t. After m mistakes on positive, its weight will be,*

$$w_n^t = (1+\beta)^m$$

Proof. Recap on the weights update rule,

$$\begin{aligned} w_n^{(t+1)} &= w_n^{(t)}(1+\beta)^{(y^{(t)}-\hat{y}^{(t)})\cdot x_n^{(t)}} \\ &= w_n^{(t)}(1+\beta)^{x_n^{(t)}} \quad (\because (y^{(t)} - \hat{y}^{(t)}) = 1 \text{ when mistake on positive}) \end{aligned} \tag{20}$$

For a relevant feature, it will always be true in each time step t, which means $x_n^{(t)} = 1$ is held in every weight update step. Therefore, by induction we can get,

$$w_n^t = (1+\beta)^m$$

Lemma 4. *Assume there is a relevant feature whose index is n, and its weight is w_n^t. Making a mistake on a positive example,*

$$w_n^{(t)} < N \quad \forall n$$

Proof. When we are making mistake on positive, it implies that the ground truth label $y^{(t)} = 1$ and our predicted label $\hat{y}^{(t)} = 0$. Let's look back to our predictor function (line 4 in Algorithm 2), it is only possible to output $\hat{y}^{(t)} = 0$ when and only when,

$$\langle \mathbf{w}^{(t)}, \mathbf{x}^{(t)} \rangle < N \tag{21}$$

From the proof of Lemma 3 we have already known $\mathbf{x}_n^{(t)}$ is always 1 (i.e. true) for relevant features. If we assume $w_n^{(t)} > N$, we can derive that $\langle \mathbf{w}^{(t)}, \mathbf{x}^{(t)} \rangle > N$, which contradicts Eq. 21. Therefore,

$$w_n^{(t)} < N \quad \forall n$$

Theorem 5. *(Mistake bound of Winnow) Let M be the total number of mistakes made by the learner, and k, N respectively be the number of relevant features and the total number of features. Assuming $\beta = 1$, then M is upper-bounded as:*

$$M < 2 + 3k(log_2 N + 1)$$

Proof. Let's define the potential function for Winnow as the sum of weights among all features,

$$\Phi^{(t)} = \sum_{n=1}^{N} w_n^{(t)} \tag{22}$$

Note that the total mistakes, m, consist of mistakes on positive examples m^+, and mistakes on negative examples m^-. Since weight update rules depend on types of mistakes, we handle each case separately to bound the potential function.

$$m = m^+ + m^-$$

We will starts with mistakes on positive examples. Following the weight update rule of Winnow in Table.1, $w_n^{(t)} = (1+\beta) w_n^{(t-1)} = 2 w_n^{(t-1)}$. It doubles the weights on relevant features where $x_n^{(t-1)} = 1$. Thus, we can derive the mathematical relation between $\Phi^{(t)}$ and $\Phi^{(t-1)}$ as follows:

$$\begin{aligned} \Phi^{(t)} &= \Phi^{(t-1)} + \sum_{n: x_n^{(t)} = 1} w_n^{(t-1)} \\ &= \Phi^{(t-1)} + \mathbf{w}^{(t-1)} \cdot \mathbf{x}^{(t-1)} \end{aligned} \tag{23}$$

Upper bound:

Recall the prediction rule in Algorithm 2 (line 5), making a mistake on a positive example means $\hat{y}^{(t)} = \mathbf{1}[\langle \mathbf{w}^{(t)}, \mathbf{x}^{(t)} \rangle > N] = 0$. In other words, $\mathbf{w}^{(t-1)} \cdot \mathbf{x}^{(t-1)} \leq N$. By substituting it into Eq. 23, the mathematical relation between $\Phi^{(t)}$ and $\Phi^{(t-1)}$ could be constrained by,

$$\Phi^{(t)} \leq \Phi^{(t-1)} + N \tag{24}$$

By induction to relate the base case using Eq. 24, the upper bound on potential from mistakes on positive examples is as follows:

$$\Phi^{(t)} \leq \Phi^{1} + m^{+}N = N + m^{+}N \tag{25}$$

Now we take a look at the mistakes on negative examples. Instead of increasing weights when making mistakes on positive examples, Eq. 23 becomes as follows:

$$\begin{aligned}\Phi^{(t)} &= \Phi^{(t-1)} - \sum_{n:x_n^{(t)}=1} \frac{1}{2} w_n^{(t-1)} \\ &= \Phi^{(t-1)} - \frac{1}{2}\mathbf{w}^{(t-1)} \cdot \mathbf{x}^{(t-1)} \end{aligned} \tag{26}$$

Similarly, making a mistake on a negative example means $\hat{y}^{(t)} = \mathbf{1}[\langle \mathbf{w}^{(t)}, \mathbf{x}^{(t)} \rangle > N] = 1$. In other words, $\mathbf{w}^{(t-1)} \cdot \mathbf{x}^{(t-1)} > N$. By substituting it into Eq. 26, the mathematical relation between $\Phi^{(t)}$ and $\Phi^{(t-1)}$ could be constrained by,

$$\Phi^{(t)} < \Phi^{(t-1)} - \frac{1}{2}N \tag{27}$$

By induction to relate the base case using Eq.27, the upper bound on potential from mistakes on positive examples is as follows:

$$\Phi^{(t)} < \Phi^{1} - \frac{1}{2}m^{-}N = N - m^{-}\frac{1}{2}N \tag{28}$$

Combining penalty from positive mistakes Eq.25 and penalty from negative mistakes Eq.28, the potential is upper-bounded by

$$\Phi^{(t)} \leq N + m^{+}N - m^{-}\frac{1}{2}N \tag{29}$$

Lower bound:

Next, as weights are initialized to 1 and sign never change during updates, one can conclude the potential is lower-bounded by,

$$0 < \Phi^{(t)} \tag{30}$$

Combine:

Then, by combining the upper bound Eq.29 and lower bound Eq.30, the potential is bounded as follows,

$$0 < \Phi^{(t)} \leq N + m^{+}N - m^{-}\frac{1}{2}N \tag{31}$$

Though we have bounded our potential function, but what we really want is to bound the total mistakes M defined below, and now we are going to derive the bound of M,

$$M = m^+ + m^- \tag{32}$$

With simple algebra on Eq. 31, we get,

$$\begin{aligned} N + m^+N - m^-\frac{1}{2}N > 0 \\ N + m^+N > m^-\frac{N}{2} \quad (\because \text{moving items}) \\ 2 + 2m^+ > m^- \end{aligned} \tag{33}$$

From Lemma 3, if m^+ mistakes are required for doing correct positive classification, and assuming $\beta = 1$, we have,

$$w_n^t = (1+\beta)^m = 2^{(m^+-1)} \tag{34}$$

Combining Lemma 4, we have, for a single relevant feature n,

$$\begin{aligned} w_n^t = 2^{(m^+-1)} < N \\ 2^{(m^+-1)} < N \\ m^+ - 1 < \log_2 N \quad (\because \log 2 \text{ base on both sides}) \\ m^+ < \log_2 N + 1 \quad (\because \text{algebra}) \end{aligned} \tag{35}$$

For k relevant features, we will have,

$$m^+ < k(\log_2 N + 1) \tag{36}$$

With Eq. 33, we can bound the mistakes on negative m^- by m^+,

$$\begin{aligned} m^- &< 2 + 2m^+ \\ &= 2 + 2k(\log_2 N + 1) \end{aligned} \tag{37}$$

So the total mistakes M, with k relevant features and N totoal number of features,

$$\begin{aligned} M &= m^- + m^+ \\ &< 2 + 2k(\log_2 N + 1) + k(\log_2 N + 1) \\ &= 2 + 3k(\log_2 N + 1) \end{aligned} \tag{38}$$

2.3 Conclusion

Online Linear Classification problem aims to learn a good linear classifier over time, by feeding correct classification answer from side-by-side experts. It is expected to converge to the performance of the expert ultimately. Therefore, it is suitable for situation where data volume is too huge to train in one batch or the classification criteria is evolving. To solve these problem, we introduce two online linear classification algorithms called Perceptron and Winnow. Both of them involve a linear hyperplane, while Perceptron predicts with sign function and Winnow predicts with logical OR function based on relevant features. Noteworthy, their number of mistakes can be bounded when assuming linear separability of data.

3 Appendix

3.1 History of Perceptron Algorithm

Initial wave of excitement:

According an article published by The New Yorker in 1958 [5], people were excited by how powerful this perceptron algorithm could be. It was successfully simulated a new electronic brain on IBM 704 computer by Dr. Frank Rosenblatt, and later was implemented as custom hardware consisting of 400 photocells for vision applications [4]. Dr. Rosenblatt described it as the first non-biological object which is able to interact with its environments and to compose concepts which have not been accessible for a human agent. It essentially tells the different between a dog and a cat without the ability to locate the dog relative to the position of the cat. Dr. Rosenblatt was so positive about its promising future that it may be applicable for space exploration [7].

Famous XOR problem:

Then the excitement was hitted by the AI winter. In the book written by Marvin Minsky and Seymour Papert and published in 1969 [6], major limitations for this perceptron algorithm were pointed out. One of the most famous counter-examples is the XOR problem shown in Figure 1. Since the data is non-separable, a single layer perceptron can't come in place.

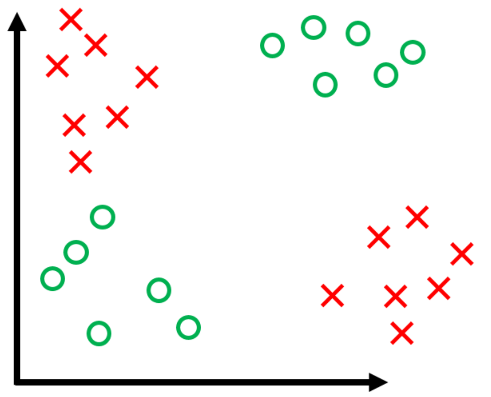

Figure 1: Illustration of a non-linearly separable data set

3.2 Comparision between Perceptron and Winnow

From source [1], we can do comparision between Perceptron and Winnow algorithm by the total number of mistakes along trials. In Figure 2, N represents the total number of variables / features, and we fix $k = 20$ (i.e. 20 relevant features) in the experiments. The graph shows how the number of mistakes are made in different trials.

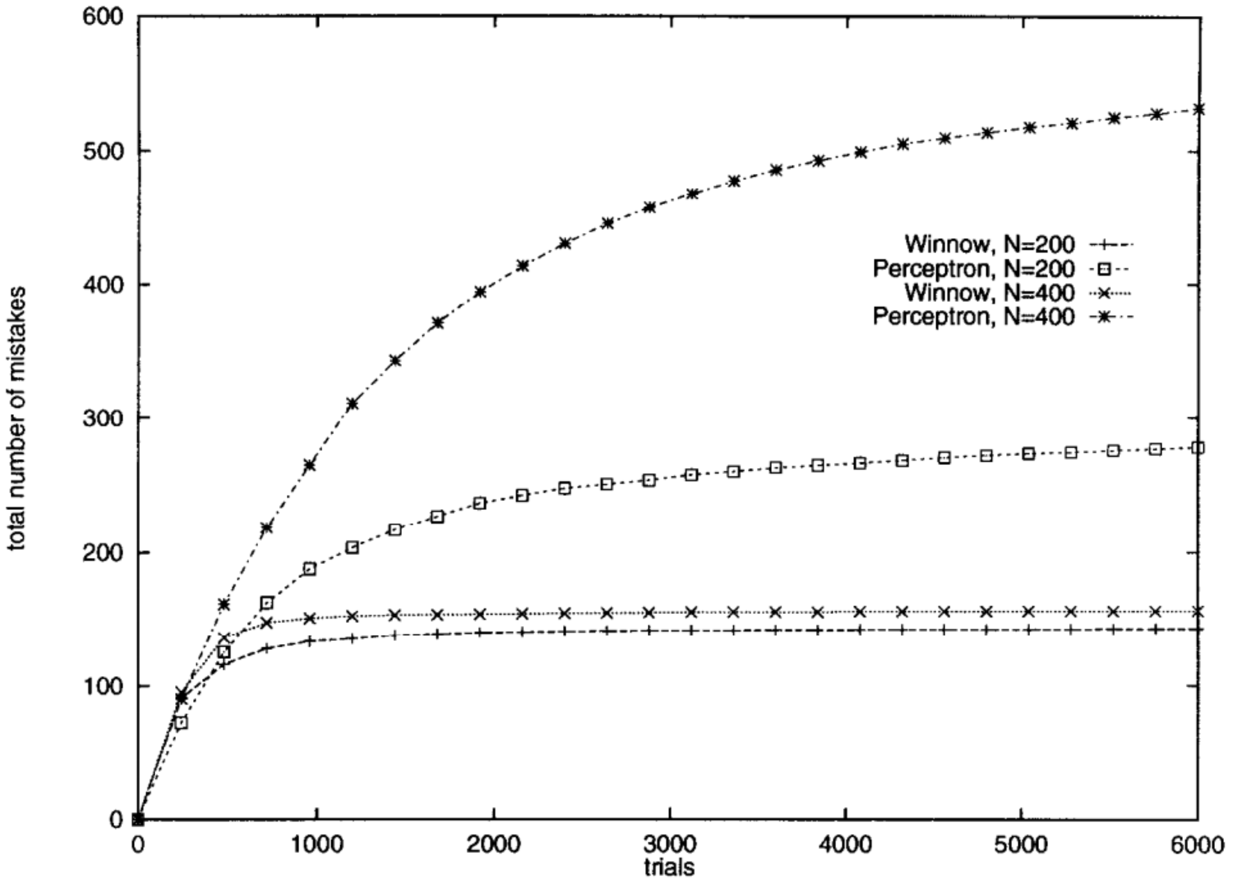

Figure 2: Cumulative mistake counts for Winnow and Perceptron algorithm

We notice that Perceptron algorithm is more sensitive to the number of features N, if we compared across $N = 200$ and $N = 400$. However, this change has little effect on Winnow. This may because Winnow algorithm is making prediction on relevant features only. Also, we can clearly notice a convergence trend for both algorithms in this graph, which matches our conclusion in Sec. 2.3.

References

[1] P. A. J. Kivinen, M.K. Warmuth. The perceptron algorithm versus winnow: linear versus logarithmic mistake bounds when few input variables are relevant. *ELSEVIER*, 1997.

[2] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 68–77, 1987.

[3] N. Littlestone, M. K. Warmuth, et al. *The weighted majority algorithm.*

[4] H. Mason, D. Stewart, and B. Gill. A neural networks deep dive. *IBM Developer.*

[5] H. Mason, D. Stewart, and B. Gill. Rival. *The New Yorker.*

[6] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry.* MIT Press, 1969.

[7] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.